A System Design of Tight Physical Integration for Large-scale Vehicular Network Emulation

Arata Kato[†], Mineo Takai[*,§], Susumu Ishihara[‡]
[†]Graduate School of Integrated Science and Technology, Shizuoka University, Hamamatsu, Japan
[*]Graduate School of Information Science and Technology, Osaka University, Osaka, Japan
[§]Computer Science Department, University of California, Los Angeles, Los Angeles, CA
[‡]College of Engineering, Academic Institute, Shizuoka University, Hamamatsu, Japan
Email: {kato.arata.17, ishihara.susumu}@shizuoka.ac.jp, mineo@ieee.org

***Abstract*—Network emulation techniques are helpful to evaluate the operation and the performance of applications and protocols on network systems including vehicular ad-hoc network (VANET). We have proposed a wireless network emulation environment using a wireless network tap device (wtap80211), a virtual wireless network device and provides interfaces that enables a network simulator to capture real IEEE 802.11 frames, transmission control information such as transmission power and frequency, and reception status information such as Receive Signal Strength Indicator (RSSI) and Basic Service Set (BSS). Therefore, with the emulation framework, a network simulator can imitate radio propagation and the mobility of network nodes. On the other hand, to fully exploit the emulation framework, monitoring and controlling a network system is important. Since the topology of VANET always change due to the mobility of vehicles, we need to oversee all network nodes to grasp the reachability of beacons sent by vehicles. In this paper, we design a network monitoring interface for a large-scale wireless network system focusing on V2V wireless networks. using wtap80211-based wireless network emulation framework and introduce mechanisms that improve the performance of network monitoring.**

***Index Terms*—VANET monitoring, wireless network emulation, wireless network tap device.**

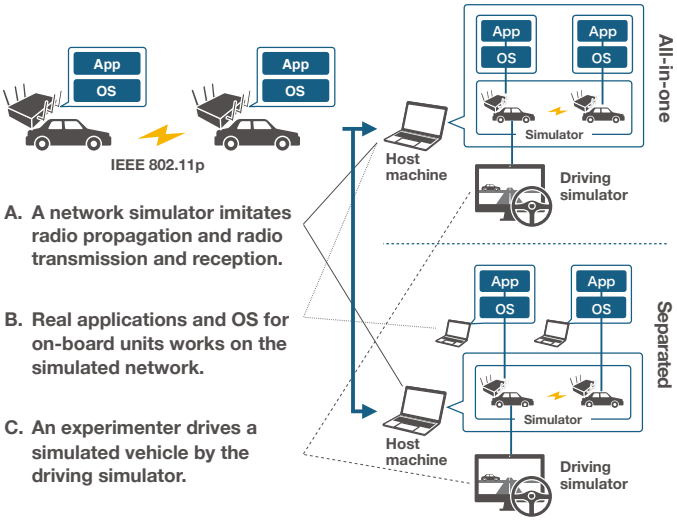


Fig. 1: Examples of vehicular network emulation.

I. Introduction

Wireless network emulation is useful for developers and researchers to evaluate the operation and performance of wireless network systems with real applications and operating systems without the construction of a physical network environment. For evaluating the behavior and performance of vehicular network (VANET) systems and applications, wireless network emulation is more suitable than experimentation because experimentation is hard to repeat with the same condition and to apply to large-scale networks such as a VANET at a city center.

Fig. 1 shows an example of vehicle-to-vehicle (V2V) communication emulation. In **Fig. 1**, an operating system and network applications work on a virtual wireless network environment in which a network simulator imitates the behavior of wireless communication environments such as radio propagation and the mobility of communication-enabled virtual vehicles. A virtual vehicle moves in the network simulation environment according to the output of a driving simulator. Wireless network emulation enables to evaluate the performance of vehicle-to-everything (V2X) transmission management protocols, such as ETSI ITS-G5 DCC and Adaptive Traffic Beacon protocol (ATB) [1] using the virtual wireless network and real application software and operating system.

In [2], we proposed a wireless network emulation framework using the wireless network tap device (*wtap80211*). We implemented a wireless network tap device as a virtual wireless LAN device driver on Linux operating system. The device software communicates with a Linux kernel and exchanges real IEEE 802.11 frames and information to manage wireless LAN communications such as received signal strength indication (RSSI), transmission power and frequency, basic service set (BSS) with wireless network emulation.

To fully exploit the emulation framework, monitoring and controlling the environment is indispensable for validating the operation and performance of the wireless network system. The emulation of a wireless network system needs to control the emulated network nodes to work according to the emulation scenario correctly. In particular, in vehicular networks, since V2V communication is broadcast-based communication because network nodes always move, it is necessary to watch

978-1-5386-9151-9/19/$31.00 ©2019 IEEE

all communication-enabled vehicles to monitor the connectivity of the vehicles because the link condition changes. In addition, application programs on vehicles generate messages according to their current situation. Therefore, the tight integration of emulated physical condition of network nodes, links, and the behavior of real software on real hardware is required.

On the other hand, visualizing and validating network statistics and the operation state of network nodes sometimes are burdensome for developers and researchers because they require to spend extra effort on developing monitoring tools not relevant to their work. In general, analysis of many log messages of emulation is generally complicated, and the monitoring tools require more machine resources as the number of emulated network nodes increases.

The popular way to store the results of emulation is to write them to a log file on a secondary storage such as a solid state drive. The overhead of filesystem APIs with the accesses to the secondary storage is, however, high because the APIs execute many system calls and require to copy data between the user space and the kernel space. Therefore logging using the filesystem APIs degrades the performance of the network system. The increase of machine loads deteriorates the adequacy of the emulation results by the observer effect, that means observing processes in running cause negative impact on the behavior of the processes.

Thus, in this paper, we design a light-weight wireless network monitoring system based on wtap80211-based wireless network emulation framework. We introduce logging mechanisms that write/read log messages to/from random access memory (RAM). The mechanisms do not access secondary storages and do not require to copy log messages between the user space and the kernel space. Therefore, the mechanisms enable to monitor many wireless network nodes working on the emulation framework without degrading the system performance because the mechanisms reduce the number of system calls compared with a case using secondary storages.

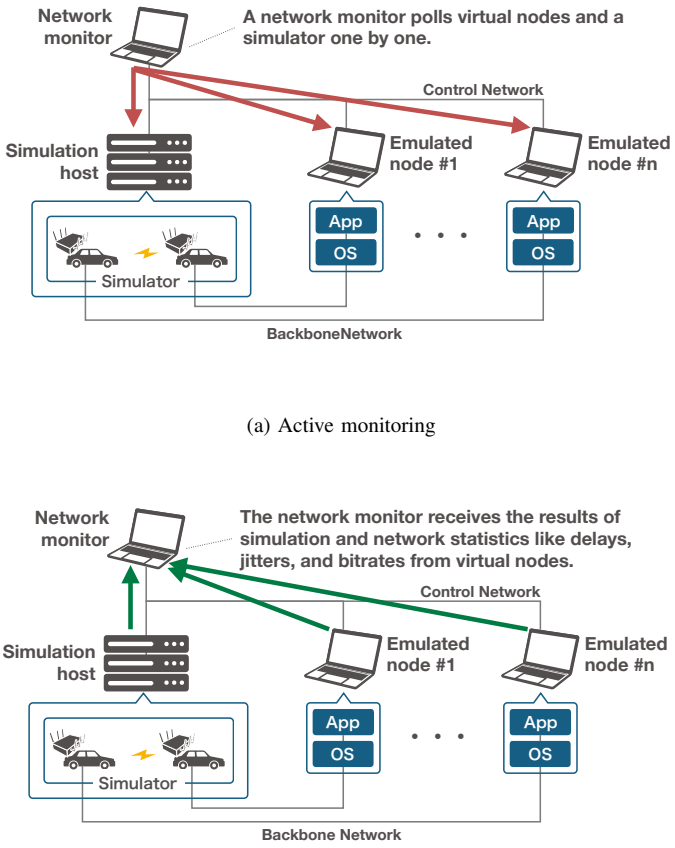
The remainder of this paper is structured as follows: Section II presents the related work. Section III introduces the wtap80211-based wireless network emulation framework. Section IV describes the design and the mechanisms of the wireless network monitoring system with the emulation framework. Section V concludes the paper with future work.

II. Related work

The common issue of network monitoring across network emulation and experiments is the difficulty of control of network nodes and treatment of a large number of log messages. In this section, we describe the features and issues of existing work that focuses on network monitoring of large-scale emulated or real networks. We classify the existing work into two categories: active monitoring and passive monitoring. Then, we present requirements for VANET monitoring in network emulation.

A. Active monitoring

Active monitoring refers to that a network monitor oversees the operation of network nodes by polling them. **Fig. 2a** shows an example of active monitoring in a network emulation environment. The network emulation environment consists of a network simulator, a network monitor, and network nodes. A backbone network is used for data communication among network nodes and a network simulator. A control network is used for that the network monitor oversees network nodes and collects the variation of system resource of each network node and the progress of the emulation.



(a) Active monitoring

(b) Passive monitoring

Fig. 2: Examples of active and passive monitoring

OpenNetMon [5] focuses on monitoring in software-defined networks. OpenNetMon configures OpenFlow switches with forwarding and filtering rules and forwards packet flows to multiple traffic monitoring systems, which calculate network statistics such as bitrates, delays, jitters, and packet loss. However, the workload of network monitoring increases as virtual nodes increases because OpenNetMon collects network statistics by polling the switches respectively and holds log messages in a file.

WiNeTestEr [6] is a wireless network testbed that specializes in channel emulation. WiNeTestEr's network monitor oversees wireless network nodes as well as FPGA boards for emulating channel conditions. The network monitor collects experiment results from each network node and records the results on a MySQL database server. The authors implemented

TABLE I: Comparison of existing monitoring methods

	Method	How to monitor	How to manage
OpenNetMon	Active	OpenFlow switch APIs	Local storage
WiNeTesEr	Active	Original library	MySQL database
DevoFlow	Passive	Packet sampling	Local storage
FlowMonitor	Passive	ns-3 APIs	RAM

a logging library which records log messages of all software modules of WiNeTestEr. The logging library outputs log messages to the standard output/error streams (stdout/stderr) and a log file. Therefore, the logging mechanisms have negative impacts on the performance of emulation as Ahmed et al. [7] indicates that the performance of a MySQL server depends on machine resources, selection of operating system, and tuning of the kernel. Additionally, it takes high loads to write enormous log messages to the standard output/error streams and a file.

B. Passive monitoring

Passive monitoring refers to that a network monitor passively captures data traffic between network nodes or analyzes reports of the simulation or experiment results. **Fig. 2b** shows an example of passive monitoring on a emulated network. In **Fig. 2b**, a network monitor knows the operation state of network nodes and the progress of emulation based on log messages of the operating system of the node and simulation results such as the number of collisions of signals and the path of moving network nodes. Passive monitoring does not require many system resources and network resources, and the workload of a network monitor is small because polling is not used to monitor network nodes.

DevoFlow [8], a modification of the OpenFlow model, improves the efficiency of network statistics collection by sampling data packets. DevoFlow randomly chooses data packets at a rate of 1/1000 packets and sends network statistics calculated based on the packets to a network monitor using sFlow [9], the standard for packet exporting. Therefore, DevoFlow reduces the load of monitoring, however, it is difficult to monitor network nodes continuously.

FlowMonitor [4], a flow monitoring module for ns-3 [11], aggregates simulation results into XML trees and stores the aggregated results on random access memory. However, it is difficult to apply logging systems such as sampling-based logging to wireless network emulation because the logging system is realized by restricting the number of information sources. Wireless network monitoring, especially VANET monitoring, requires to monitor not only network statistics such as delays and packet loss but also channel condition and the position of wireless network nodes.

C. Requirements for VANET monitoring in network emulation

Active monitoring enables us to know the operation of network nodes in real-time, While passive monitoring makes it possible to monitor network nodes without degrading network performance. However, active monitoring requires many network and system resources. Passive monitoring has the limitation of the kinds of collective information. On the other hand, some popular open-source network monitoring systems such as Nagios [12], Zabbix [13], and OpenNMS [14] supports both active and passive monitoring. However, they require that remote servers that respond requests from a network monitor must be installed to network nodes respectively.

In VANET emulation, it is necessary to imitate traffic flows of vehicles as well as characteristics of wireless communication like radio propagation. Moreover, since V2V communication is broadcast-based communication and the topology of VANET always change because vehicles frequently move, the scalability of VANET emulation is essential. For example, the performance evaluation of V2V communication management protocols such as ETSI DCC and ATB requires macro-perspective monitoring that oversees all communication-enabled vehicles to grasp the connectivity of the vehicles.

On the other hand, the performance evaluation of V2V systems such as Advanced Driver Assistance System (ADAS) requires micro-perspective monitoring that watches specific vehicles in a specific area like an intersection. The V2V systems depend on many kinds of information such as the sensing data and the position of vehicles, and the vehicles move cooperatively by exchanging the information. Therefore, it is essential for micro-perspective monitoring to collect information as much as possible.

Based on the discussion above, we summarize realizing an suitable network monitoring in VANET emulation follows:

1) Scalability and flexibility: Although there are some differences in scalability required for VANET monitoring, VANET monitoring should support both macro- and micro-perspective monitoring for VANET emulation in various situations.

2) Small impact: Since VANET monitoring system should manage a large amount of information, VANET monitoring system should have a small negative impact on the resources involved in VANET emulation to avoid the observer effect because it may cause anomalous behavior. For example, it is undesirable to read and write a file using filesystem I/O frequently for logging the system status. In particular, since our emulation framework can access the kernel and track the control flows in network protocol stacks, it is necessary to reduce the overhead of monitoring as much as possible.

3) Simple structure: Complicated network monitoring software sometimes makes anomalous behavior that appears when using the monitoring software. Simplifying the structure of monitoring software is useful for developers and researchers to debug and evaluate VANET and the other various network applications and reduce monitoring workload.

4) Real-time monitoring: Since the state and the structure of VANETs are always not constant, collecting feedbacks during network emulation is useful to grasp the behavior of VANET systems and detect issues and errors of the system in running.

III. WTAP80211-BASED WIRELESS NETWORK EMULATION FRAMEWORK

In this section, we introduce the overview of wtap80211-based network emulation framework.

A. Overview

wtap80211-based wireless network emulation framework consists of wireless network tap device (wtap80211), wtap80211 daemon, and a network simulator.

Fig. 3 shows the overview of the emulation framework that emulates communication between an access point and a station node. In the emulation framework, wtap80211 makes virtual wireless network interfaces in the kernel when installed. Those interfaces have the same APIs to Linux wireless subsystem as real wireless interfaces. Linux wireless subsystem is a set of kernel modules that controls and manages wireless devices and communications, concretely, Linux wireless subsystem executes processes defined in IEEE 802.11 MLME (Mac sub-Layer Management Entity). Data packets that user applications send are encapsulated into IEEE 802.11 frames by Linux wireless subsystem and the frames are handed over to wtap80211.

After receiving IEEE 802.11 frames from Linux wireless subsystem, wtap80211 transfers the frames to wtap80211 daemon program. wtap80211 daemon, a daemon process in the user space, transfers data received from wtap80211 to another user application such as a network simulator after converting the data format of them if needed.

A network simulator imitates the operation of carrier sensing and the behavior of the physical layer such as transmission and reception of signals by wireless communication equipments, radio propagation, motility of wireless network nodes, etc. Therefore, network applications running on the user space and Linux wireless subsystem works in emulation, as with real environments.

B. Wireless network tap device

A wireless network tap device (*wtap80211*) is a virtual wireless network driver that supports IEEE 802.11a/b/g/n/ac and IEEE 802.11p/ad and works as an interface between Linux wireless subsystem and a user application. wtap80211 has a function to exchange IEEE 802.11 frames and transmission control and reception status information such as transmission power, Basic Service Set (BSS), and Receive Signal Strength Indicator (RSSI) with Linux wireless subsystem via kernel APIs.

In addition, wtap80211 has another data path with user applications using Netlink [10], which is one of socket interfaces and used for inter-process communication or communication between a user application and a kernel module. Therefore, wtap80211 transfers a Netlink message that includes transmission control information such as transmission power and frequency when receiving a control command from Linux wireless subsystem, and wtap80211 notifies the change of reception status such as RSSI to Linux wireless subsystem when receiving the simulation results such as RSSI and BSS information.

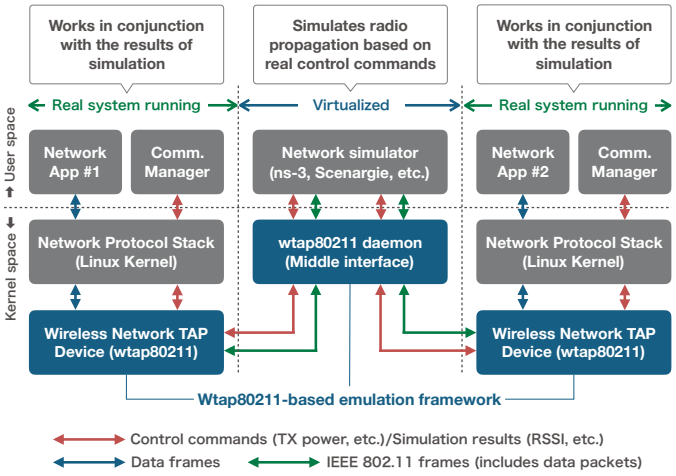


Fig. 3: Detail of the architecture of our emulation framework.

wtap80211 and wtap80211 daemon exchange Netlink messages including IEEE 802.11 frames or parameters to control wireless communication and devices. A Netlink message including IEEE 802.11 frames consists of IEEE 802.11 frame and either transmission control information or reception status information. The transmission control information includes transmission power, frequency, bitrate, and beacon interval. The reception status information includes RSSI, reception time, and the number of streams if either HT (High Throughtput) or VHT (Very High Throughtput) mode is used.

A network message including parameters to control wireless communication and devices contains Basic Service Set (BSS) information, channel switch data, configuration of the devices. The channel switch data include the new channel band and frequency to switch and timestamp of the data. The configuration of the devices includes the status of hardware switches, minimum threshold of transmission power, channel frequency to tune in, and powersave timeout. When Linux wireless subsystem or wtap80211 changed the parameters, wtap80211 notifies user applications of the change of the parameters via wtap80211 daemon.

C. wtap80211 daemon

wtap80211 daemon (*wtap80211d*) is a daemon application that provides a function of data exchange between user applications and wtap80211. The daemon provides user applications with multiple interfaces to make a connection to wtap80211. Although network simulators that support network emulation provide user applications with interfaces to forward packet flows, the specifications of the interfaces are different in each network simulator. Thus, the interfaces that wtap80211 daemon provides user applications support socket APIs, filesystem I/O and Netlink APIs.

The daemon decapsulates Netlink messages received from wtap80211 and the content of each Netlink message to a user application via the APIs. Since wtap80211 encapsulates IEEE 802.11 frames and control messages in Netlink messages and sends them to wtap80211 daemon,

IV. Monitoring System for Large-Scale VANET Emulation

In this section, we present the problems in monitoring with wtap80211-based emulation framework and design an on-memory logging system for wireless network emulation. For reducing the overhead in monitoring such as file accesses as much as possible, the logging system stores a large number of log messages and simulation results in a ring buffer on multiple shared memory blocks.

A. Problems of monitoring in wtap80211-based emulation framework

Our emulation framework is susceptible to the observer effect. We have confirmed that network applications, protocol stacks including wireless LAN modules in an operating system, and virtual wireless devices works in real time in [2]. In the emulation framework, wtap80211 daemon collects many kinds of information of use in the analysis of the behavior of network nodes and the progress of network emulation. Although the feature enables developers to simplify debugging and the error detection of the software in running, the processing load tends to concentrate at wtap80211 daemon.

Since our wireless network emulation framework accesses data structures in the kernel to capture IEEE 802.11 frames and collect information relevant to wireless LAN communication, resource-hungry logging like dumping into a physical disk deteriorates the performance of wtap80211 daemon and makes the kernel be panic and crash at the worst. In general, the frequent calls of the system call functions such as `write()` and `read()` deteriorates the performance of processes because such functions copy the content of memory between the user space and the kernel space. Thus, reducing the number of calls of the system call functions is effective to improve the performance of wtap80211 daemon.

B. System overview

Fig. 4 shows the system overview of the designed monitoring system. The monitoring system is installed to each host with emulated network interfaces and consists of wtap80211 daemon, a logger process, and a ring buffer shared between the daemon program and the logger program.

The data structure of buffer memory is typically designed as a queue or a ring buffer. Since a ring buffer is fixed-size buffer linked end-to-end, it reduces the memory consumption if wtap80211 daemon writes enormous log messages to the buffer. In addition, since the daemon can overwrite old log messages with new ones, the daemon program can keeps writing log messages to the ring buffer if all buffers are full of messages. In contrast, a queue is FIFO (Fist-In-Fast-Out) buffer. If the size of the queue is fixed, the daemon program is blocked when the queue is full of log messages. If the queue size is variable, the daemon can write log messages even when the queue is full of other log messages. However, the variable-size queue increases the memory consumption as the size of log messages increases. Therefore, we use a ring buffer to store log messages in the designed monitoring system.

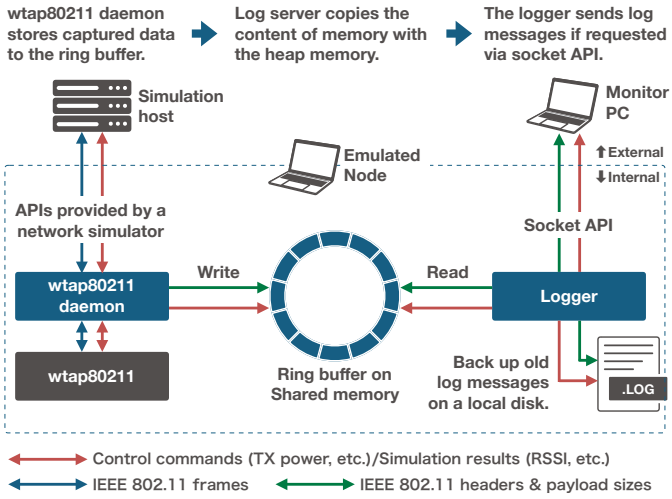


Fig. 4: Overview of the designed monitoring system.

The logger runs as a child process of wtap80211 daemon. Since the child process has its memory fields independent of the memory field owned by the parent process, the logger can keep running even if wtap80211 daemon crashes and notify an external monitor host of that the daemon has crashed.

wtap80211 daemon and the logger share a ring buffer composed of multiple shared memory blocks for inter-process communication between them. In the monitoring system, storing and passing log data are processed on the ring buffer only. wtap80211 daemon stores log messages including captured IEEE 802.11 header and payload size, information relevant to wireless LAN communication and the simulation results to the ring buffer. The daemon should not store the content of the payload field of IEEE 802.11 data frames for saving the capacity of the ring buffer.

The logger allocates heap memory fields, and copies log messages stored in each shared memory block of the ring buffer into the heap memory fields periodically, and backs up log messages on a local disk in order of the timestamp. The backup operations are performed in separate multiple threads. However, if the ring buffer or the heap memory fields are full of log messages, the logger immediately backs up the content of the heap memory field to the local disk.

The logger sends log messages via socket APIs if an external monitor host requests the messages. However, the logger does not provide the database function because maintaining a database during network emulation involves many system calls. Since a system call leads to the context switches and copying the content of memory between the user space and the kernel space, many system calls deteriorate the system performance. In addition, since the logger is installed on each emulated network node, we need to manage multiple databases if each logger uses a database.

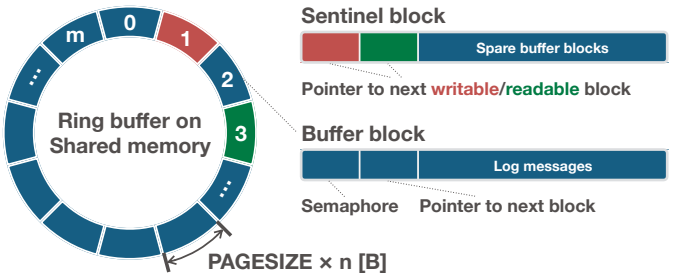


Fig. 5: Composition of the ring buffer with shared memory blocks.

C. Mechanisms of on-memory logging with wtap80211 daemon

Fig. 5 shows the detail of the ring buffer. The ring buffer consists of buffer blocks and a sentinel block. Since the logger process is tolerant of a failure of wtap80211 daemon, the logger manages the ring buffer. A Buffer block includes a semaphore, a pointer to the next block, and a storage field for log messages. The sentinel block includes pointers to the next writable or readable block and additional buffer blocks.

We allocate a small memory block for each shared memory block except for the sentinel block to quicken the memory I/O access. The buffer block size, 4 KiB, is determined as the same as the typical page size of memory on general 64-bit computers. The reason why we do not allocate a large sequential block is to shorten the blocking time to synchronize the content of the buffer blocks with the heap memory of the logger. If a large amount of buffer block is allocated at once, the writing operation is blocked every time the semaphore blocks the operation of reading log messages from the buffer block. While the logger is copying the content of the buffer block, the blocking time becomes longer as the amount of logging data increases. Since wtap80211 daemon records enormous log messages that include IEEE 802.11 headers and payload size, the increase of the blocking time is not ignorable.

The sentinel block has multiple additional buffer blocks to reduce the possibility of overwriting the content of buffer blocks in the ring buffer. wtap80211 daemon writes log messages to the additional buffer blocks if all buffer blocks on the ring buffer are not writable because all buffer blocks are full of log messages or the logger locks the semaphore to read log messages. The logger inserts additional buffer blocks at the tail of the ring buffer when the writing process catches up the reading process, or the existing buffer blocks are full of messages.

V. Conclusions

For large-scale vehicular network emulation, we designed a system for tight physical integration of network nodes, network simulator, and traffic simulator like a driving simulator. We discussed issues in the monitoring of large-scale emulated wireless network including VANET and presented the design of an light-weight on-memory logging system using a wireless network tap device and wtap80211 daemon. The logging system stores many log messages of an operating system and simulation results on a ring buffer on shared memory. Therefore, the logging system reduces the overhead of logging and the negative impact on the behavior of monitored processes. In addition, other techniques such as tuning the kernel parameters can be applied for further improvement of memory access performance.

In this paper, we focus on monitoring network nodes and a network simulator. However, controlling network nodes connected to the network simulator is also required in practice. In future work, we expand the monitoring system to support to control network nodes and the network simulator, and implement the designed system in the Linux operating system and evaluate the performance of the system.

References

[1] C. Sommer, O. K. Tonguz and F. Dressler, “Adaptive Beaconing for Delay-sensitive and Congestion-aware Traffic Information Systems”, In Proceedings of 2010 IEEE Vehicular Networking Conference, Jersey City, NJ, pp. 1–8, 2010.
[2] A. Kato, M. Takai, S. Ishihara, “Design and implementation of a wireless network tap device for IEEE 802.11 wireless network emulation”, In Proceedings of 2017 Tenth International Conference on Mobile Computing and Ubiquitous Network (ICMU), Toyama, Japan, pp. 1–6, 2017.
[3] T. Rakotoarivelo, M. Ott, G. Jourjon, and I. Seskar, “OMF: a control and management framework for networking testbeds,” ACM SIGOPS Operating Systems Review, Vol. 43, Issue 4, pp. 54–59, January 2010.
[4] G. Carneiro, P. Fortuna, and M. Ricardo, “FlowMonitor - a network monitoring framework for the Network Simulator 3 (NS-3),” In Proceedings of the 4th International ICST Conference on Performance Evaluation Methodologies and Tools, Pisa, Italy, pp. 1–10, 2009.
[5] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, “OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks,” 2014 IEEE Network Operations and Management Symposium (NOMS), Krakow, pp. 1–8, 2014.
[6] J. D. Deshay, K. S. Subramani, N. Mahabeleshwar, E. Nourbakhsh, B. McMillin, B. Banerjee, R. Prakash, Y. Du, P. Huang, T. Xi, Y. You, J. D. Camp, P. Gui, D. Rajan, and J. Chen, “Wireless Networking Testbed and Emulator (WiNeTestEr),” Computer Communications, Vol. 73, Part A, pp. 99–107, Elsevier, 2016.
[7] M. Ahmed, M. M. Uddin, Md. S. Azad, and S. Haseeb, “ MySQL Performance Analsys on a Limited Resource Server: Fedora vs. Ubuntu Linux”, In Proceedings of the 2010 Spring Simulation Multiconference, Orlando, FL, pp. 1–7, 2010.
[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “DevoFlow: Scaling Flow Management for High-Performance Networks”, In Proceedings of the ACM SIGCOMM 2011 Conference, Toronto, Canada, pp. 254–264, 2011.
[9] P. Phaal, S. Panchen, and N. McKee, “InMon Corporation’s sFlow: A Method for Monitoring Traffic in Switched and Routed Networks”, RFC 3176, IETF, 2001.
[10] J. Salim, H. Khosravi, A. Kleen, and A. Kuznetsov, “Linux Netlink as an IP Services Protocol”, RFC 3549, IETF, 2003.
[11] nsnam, “ns-3”, https://www.nsnam.org (Accessed at 11/24/2018).
[12] Nagios Enterprises, LLC., “Nagios: The Industry Standard in IT Infrastructure Monitoring”, https://www.nagios.org (Accessed at 11/24/2018).
[13] Zabbix LLC., “Zabbix” https://www.zabbix.com (Accessed at 11/24/2018).
[14] The OpenNMS Group, Inc. https://www.opennms.org/en (Accessed at 11/24/2018).